Algebraic Framework for Synchronous Language Semantics

Daniel Gaffé, Annie Ressouche

Algebraic Framework for Synchronous Language Semantics

Daniel Gaffé
Inria-sam and LEAT Sophia Antipolis University, CNRS
2004 route des lucioles BP 93
06902 Sophia Antipolis Cedex France
Email: daniel.gaffe@unice.fr

Annie Ressouche
Inria-sam
2004 route des lucioles BP 93
06902 Sophia Antipolis Cedex France
Email: annie.ressouche@inria.fr

***Abstract*—In this article, we study several relevant algebraic frameworks to define synchronous language semantics. Synchronous languages are quite dedicated to design critical embedded applications. Thus, verification and compilation is challenging and should rely on mathematical semantics. We study multi-valued algebras as foundation for semantics definition and we show that a 4-valued algebra with a bilattice structure is well suited to our concern. With this approach we can define semantics offering both the generation of models where verification techniques apply, and *separated* compilation means.**

***Keywords*—synchronous languages, synchrony paradigm, Boolean algebra, multi-valued algebras**

I. Introduction

Nowadays *Synchronous languages* quite answer the increasing need for reliable critical software, specially in the design of embedded systems. The synchrony paradigm is a mathematically sound foundation for the design of concurrent and deterministic applications. Synchronous program semantics build formal models and the definition of the theoretical framework used to support them is an important challenge according to the targeted goals (verification, compilation, etc..).

A. Fundamentals of Synchrony

Synchronous languages rely on the *synchronous hypothesis* which assumes a discrete time scale, made of logical instants corresponding to reactions of the system. All the events concerned by a reaction are simultaneous : input events as well as the triggered output events. As a consequence, a reaction can be considered as instantaneous (a reaction takes no time), there are no concurrent partial reactions therefore, determinism can be ensured. Indeed, real time is abstracted by a logical time. Communications between actors are also supposed to be instantaneous. The only communication and synchronization means between sub programs are signals or flows.

B. Synchronous Languages Semantics

A fundamental concept of the synchronous paradigm is the notion of *reaction*. Indeed, synchronous programs react to input events by emitting output events and reaching a new state. A program can be viewed as a possibly infinite sequence of reactions.

Basically, events are signals in a current environment (external and internal) with a status (present or absent). Semantics formally compute an output environment according to an input one for each reaction. Let us consider $\mathscr{S}$ a set of signals, an environment E is a function who defines a Boolean status (0 means absent, 1 means present) for each signal: E: $\mathscr{S} \mapsto \mathbb{B}$. Then, semantics defines rewriting of the form $P \xrightarrow[I]{O} \delta(P)$ where P is a synchronous program, I an input environment and O the resulting output environment, $\delta(P)$ is the derivative of P, i.e the new program that will react to the next input environment. The reaction O_1,O_2,...,O_n,... to an input environment sequence I_1,I_2,...,I_n,... results from the sequence of transitions:

$$P \xrightarrow[I]{O} P_1 \xrightarrow[I_1]{O_1} P_2 .....P_n \xrightarrow[I_n]{O_n} P_{n+1} ....$$

Each transition $P \xrightarrow[I]{O} \delta(P)$ is structurally computed from the body instruction of the program according to rewriting rules defined for each construct of the language. These rules define the computation of output environment from input ones. All the synchronous languages have an operator (*emit* or a similar one), to change the status of signal from absent to present. A *logical coherence law* helps us to assign status to signals. This law says that: " a signal **S** is present in a reaction if and only if an **emit S** statement is executed" [1]. Then rewriting rules for operators formalize this signal coherence law.

C. Algebras as Mathematical Framework

An efficient means to express these rewriting rules aforementioned is to use the Structural Operational Semantics (S.O.S) style defined by G.Plotkin [2]. S.O.S rules are deduction rules of the shape:

$$\frac{Premiss(1)\ Premiss(2)\ ...\ Premiss(n)}{Conclusion}$$

meaning that: $\bigwedge_{i=1}^{i=n} Premiss(i) \Rightarrow Conclusion$. For instance, if we consider the parallel ($\|$) operator of *Esterel* language [1], the shape of its rule is:

$$\frac{p_1 \xrightarrow[I]{O_1} p_1', p_2 \xrightarrow[I]{O_2} p_2'}{p_1 \parallel p_2 \xrightarrow[I]{O_1 \uplus O_2} p_1' \parallel p_2'}$$

This example is representative of synchronous language parallel rules. We can see that the rule performs a specific

operation of union of environments ($O_1 \uplus O_2$) which "unifies" the information concerning the respective status of signals in both O_1 and O_2 output environments. If the status of S in O_1 is 0 (absent) and is 1 in O_2 (present), in $O_1 \uplus O_2$, the status of S should be 1 (in a Boolean consideration of signal status).

Hence, to define semantics rules formally, we need to give an algebraic framework to represent both signal status and operations on them and on environments. A natural first approach considers a Boolean algebra with $\wedge$, $\vee$ and $\neg$ operators. Then, starting from the fact that all signals (except input signals present in the reaction) have status 0 in the initial environment, the $O_1 \uplus O_2$ operation turns out to be the $\vee$ operation on respective signal status in O_1 and O_2.

According to synchrony paradigm, correct programs should be both *reactive* and *deterministic*. Reactivity means that a synchronous program must always react to any input event sequence, possibly in doing nothing. Determinism means that computations are reproducible and then a program always yields the same output event sequence in reaction to an input sequence. Such programs are called *logically correct* (see [1]). On another hand, a challenging phenomenon that synchronous language semantics have to deal with is the notion of *causality*. Causality means that for each event generated in a reaction, there is a causal chain of events leading to this generation. No causal loop may occur.

To take into account these two aspects of synchrony paradigm, G. Berry [1] introduced *constructive* semantics for the Esterel language. But, in these semantics the mathematical framework is no more a Boolean algebra but a ternary algebra. Then, 4-valued algebras have been considered to check causality with fixpoint iterative techniques [3] or to get separated compilation means relying on semantics definition [4].

This article studies different algebraic frameworks useful to define synchronous languages semantics. We answer the question: which algebra is well suited to define the different semantics provided for synchronous languages? Indeed, we are designing a toolkit CLEM [5] around a new synchronous language supporting a separated compilation. During this activity, we realized that a 4-valued algebra is a necessary foundation for a semantics allowing both verification and compilation. The major contribution of this paper is to show that *billatice* structure for 4-valued algebra is appropriate and supplies the correct context to define the semantics we want.

This article is organized as follows: next section (section II) introduces 3-valued and 4-valued algebras and studies their respective properties. We particularly highlight a specific 4-valued algebra (called Algebra5 in this paper) which provides us with a semantics that we can rely both on performing a separated compilation and getting verification ability. In section III, we show that bilattice structure of algebras allows to deduce a nice encoding of 4-valued algebras into Boolean pairs, in order to implement semantics rules as compilation technique. Section IV compares our approach to others. Finally, section V concludes and discusses the application of this study we have done.

II. Multi-valued Algebras for Synchronous Language Semantics

Behavioral semantics was first defined for Esterel. It is specified using S.O.S rules and considers a Boolean algebra to represent signal status. It defines globally each reaction. The values of output environment and derivative are solutions of fixpoint equations resulting from signal coherence law application and instantaneous information exchange between concurrent statements. But, the equations have non monotonic operators (the negative one essential to define local signal rules). Thus existence and computation of fixpoints cannot be ensured. As a consequence, it is ineffective. Moreover, it cannot characterize logically correct programs! It just gives a formal definition of program behaviors. In complement, an operational semantics is needed to compile programs and the equivalence between the two semantics has to be established.

Then, a *constructive* semantics characterizes logically correct programs and solves the causality problem. Its purpose is to replace "the idea of checking assumptions about signal status by the idea of propagating facts about signal status" [1]. A constructive version of behavioral semantics has been defined and also a constructive *circuit* semantics that translates each program into circuits. In this constructive approach, a 3-valued algebra has been used to represent signal status. On another hand, declarative synchronous language as Signal [6] also considers a ternary algebra to represent signal status.

A. 3-valued Algebras

1) 3-valued Algebras as Lattices: The general multi-valued algebra are introduced by Emil Post in 1921 [7]. Concerning the synchronous languages, one has proposed to extend first the *present (*1*)* and *absent (*0*)* status of each signal with *bottom (*$\bot$*)*. Intuitively, $\bot$ represents the unknown status. Thus, a 3-valued algebra $\{\bot, 0, 1\}$ is considered to represent signal status. More generally, 3-valued algebras have been introduced several decades ago for reasoning with uncertain knowledge, mainly in Logics. In these approaches, an ordering gives them a lattice structure. Moreover, these algebras are *complete distributive lattices* according to the following definition:

Definition 1. A **lattice** is a partially ordered set (L,$\leq$) in which each pair a, b has a least upper bound ($a \vee b$) and a greatest lower bound ($a \wedge b$). L is *complete* if any subset $X \subseteq L$ has a least upper bound and a greatest lower bound. L is *distributive* if it satisfies the *distributive law* : $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ or the inverse equivalent law.

The elements $a \vee b$ and $a \wedge b$ are usually called the *meet* and *join* of a and b.

This characterization as lattices allows to consider 3-valued algebras as models for 3-valued logic. Moreover, there are many applications of these 3-valued algebras in non usual logic domain as synchronous language semantics. In Logic domain, a lot of authors are well-known to develop a 3-valued algebra. In [8], we study in length several popular 3-valued algebras and compare them with respect to their ability of providing us with suitable semantics for synchronous languages. These algebras are extensions of Boolean algebra structure. They almost all

[1] G.Berry in [1]

$\vee$	1	0	$\perp$
1	1	1	$\perp$
0	1	0	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

$\wedge$	1	0	$\perp$
1	1	0	$\perp$
0	0	0	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

$\rightarrow$	1	0	$\perp$
1	1	0	$\perp$
0	1	1	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

$\neg$	
1	0
0	1
$\perp$	$\perp$

TABLE I. OPERATOR DEFINITIONS IN BOCHVAR ALGEBRA

agree on the definition of $\neg$ operator ($\neg\perp = \perp$). They differ in the way they extend $\vee$ and $\wedge$ operators on $\perp$. In the algebras we studied, most of them extend these two operators for $\perp$ such that $1 \vee \perp = 1$ and $0 \vee \perp = \perp$; $1 \wedge \perp = \perp$ and $0 \wedge \perp = 0$. However, some authors as Bochvar [9] introduced other definitions. Bochvar algebra (sometimes called weak Kleene) considers that $\perp$ is an absorbent element for $\vee$ and $\wedge$ (see table I) . We will see in II-A2, that this algebra turns out to be the algebraic formalism used to define Signal language semantics.

In all the algebras we studied (except Lukasiewicz algebra [10]), $\perp$ the *undefined* symbol, is considered implicitly or explicitly as "$\frac{1}{2}$". So, $0 \leq \perp \leq 1$. We call this ordering "Boolean order" ($\leq_B$) in the following of this article. Concerning Lukasiewicz algebra, $\perp$ is interpreted as 2 and then the ordering $0 \leq 1 \leq \perp$ is implicit.

2) Application to the Signal Language: The **Signal** [6] synchronous language considers a 3-valued algebra to compute signal status: present, absent and $\perp$. Signal language has no global clock, contrary to others synchronous languages. Hence, signal status have a specific interpretation: $\perp$ means that the signal has no clock i.e is not defined. Present and absent have the usual meaning for signal having clock. Status computing is performed in $\mathbb{Z}/3\mathbb{Z}$ with $+1$, -1 and 0 respectively encoding present, absent and $\perp$. With this encoding, each operator of the language has a ternary equation representation:

$$
\begin{array}{lcl}
\neg a & : & -a \\
a \wedge b & : & ab(ab - a - b - 1) \\
a \vee b & : & ab(1 - a - b - ab) \\
when\ b & : & -b - b^2 \\
a\ when\ b & : & a(-b - b^2) \\
a\ default\ b & : & a + (1 - a^2)b \\
a\$1 init x0 & : & a^2 x \text{ with } x'(xnext) = a + (1 - a^2)x
\end{array}
$$

The associated algebra has particular properties:

$$
\left\{
\begin{array}{lcl}
a + a & = & -a \\
a + a + a & = & 0 \\
a^{2n} & = & a^2 \\
(1 - a^2)^2 & = &
\end{array}
\right.
\quad
\left\{
\begin{array}{lcl}
a^{2n+1} & = & a \\
a.(1 - a^2) & = & 0 \\
(f(a^2))^n & = & f(a^2)
\end{array}
\right.
$$

These properties are used to compute the clock dependencies and deduce a possible clock order.

From $\vee$, $\wedge$ and $\neg$ algebraic definition, we can deduce their truth table. It turns out that it is Bochvar algebra (see table I and [8]).

3) 3-valued Algebras as CPOs: However, totally ordered algebras and lattice structures do not always fit synchronous language semantics requirements, particularly concerning fixpoint computations. Mainly because the $\leq_B$ order does not reflect how the information about variable computation grows. In constructive semantics, a program is translated into a Boolean equation system and its least fixpoint solution determines an output environment. 3-valued algebra approach allows to characterize the operational semantics of the synchronous compiler: its goal is to transform and stabilize all $\perp$ status of internal and output signals to 0 or 1. Thus, a more appropriate 3-valued algebra to compute equation system solutions is required.

We have to wait Scott [11] to reconsider the order relation. *Scott Domains* allow partial algebraic data representation, where knowledge about the elements is ordered. The goal is to interpret the elements of such domains as pieces of information or (partial) results of a computation, where elements that are higher in the order extend the information of the elements below them in a consistent way. This theory allows to compute program meaning as the limit of a sequence of approximations and Kleene fixpoint theorem [12] gives an effective means to compute this limit. Scott domains have a least element (usually called $\perp$) and support the following order: $x \leq y$ iff either $x = \perp$ or $x = y$.

A particular Scott domain is the Boolean flat domain ($\mathbb{B}_\perp$). It is the usual Boolean set with and additional $\perp$ element. According to the order definition for Scott domains, $\perp \leq 0$ and $\perp \leq 1$. In our view (also following multi-valued logic community), we call this order relation $\leq_K$ because it characterizes the *degree of knowledge*. We can note that 0 and 1 are incomparable in this relation because they have the same weight of knowledge.

4) Application to the Esterel Language: The constructive semantics of Esterel language relies on $\mathbb{B}_\perp$ Scott domain to compile programs. But $\perp$ has not the same interpretation as in Signal language. Esterel has a global clock and at each instant of its clock, signal must be stabilized as present or absent. $\perp$ has been introduced to allow the definition of an operational semantics based on Scott domain $\mathbb{B}_\perp$, in which the computation of signal status converge for all signals. Given an initial signal environment where unknown signals have the status $\perp$, the constructive semantics computes for each statement the status of signals: if the signal is emitted in *one* instruction of the statement its status is 1 (present), otherwise if it is not emitted in *all* instructions its status is 0 (absent).

This computation of absence of signals at each instant does not allow us to rely on such a constructive semantics to achieve a separated compilation. Indeed, during a separated compilation, there could exist signals that are never emitted in a sub program (then their status is set to absent). However, one of these signals can be emitted in another sub program separately compiled and we cannot change the status in the environment because present and absent are incomparable with respect to $\leq_K$ order and they haven't an upper bound. To face this drawback, we proposed a solution [4] based on 4-valued algebra to represent signal status.

B. 4-valued algebra

4-valued logics have been first considered by Belnap [13] to represent the knowledge in Artificial Intelligence systems. In such deductive systems, the information need to fall in *true*, *false*, *uncertain* or *conflicting* truth values. Thus, 4-valued logics have four truth values: $\perp$, 0, 1 and $\top$. In section II-A, we saw that constructive semantics have the ability to take into account causality. Nevertheless, causality checking remains a

global process applied at program level. It prevents to benefit from the structural rules of the semantics to separately compile programs.

We propose to rely on a semantics that associates a 4-valued algebra equation system ($\mathscr{E}$) to each program instead of a Boolean one as traditional constructive semantics do. In each reaction, the equation system $\mathscr{E}$ helps us to compute an output environment from an input one: $\mathscr{E} : E = F(E)$. $\mathscr{E}$ is built according to semantics rules defined for each operator of the language and helps us to compute the status of signals in the environment. To express the equations defining the status of output and local signals from input and local signal status, the algebra should supply two kinds of operators. First, logical operators ($\neg$, $\boxplus$, $\boxdot$) to express the equations defined by the operator rules (for instance, the local operator rule needs the negation). Second, the algebra should also provide us with operators to compute the knowledge about signals. For instance, let us consider the parallel operator ($P_1 \parallel P_2$). Let $\mathscr{E}_1$ and $\mathscr{E}_2$ be the respective equation systems of P_1 and P_2, to build the overall equation system we must compose $\mathscr{E}_1$ and $\mathscr{E}_2$ in such a way that every output has an only solution, because the parallel operator is deterministic. To this aim, we introduce an operation of "unification" of $\mathscr{E}_1$ and $\mathscr{E}_2$. This operation, we call *Unify* ($\sqcup$), performs the unification of signal status in the global equation system. For instance, assume that signal S has an equation $S_1 = f_1(\overrightarrow{v})$ computing its status in $\mathscr{E}_1$, and an equation $S_2 = f_2(\overrightarrow{w})$ in $\mathscr{E}_2$. Then the equation system for $P_1 \parallel P_2$ will have an equation $S = f_1(\overrightarrow{v}) \sqcup f_2(\overrightarrow{w})$. Intuitively, this operation must perform the union of the information concerning S status respectively in P_1 and P_2. Then, the semantics computes the unique least fixpoint $E = F(E)$ in the 4-valued algebra considered. To ensure that least fixpoints exist and can be computed, we need a 4-valued algebra with operators making F monotonic. To this aim, we also consider the symmetric operator of $\sqcup$, called $\sqcap$.

On another hand, we know that we cannot totally rely on any semantics to perform a separated compilation because synchronous language semantics cannot be both modular and causal [14]. Since in Esterel circuit semantics, the causality is checked by sorting the Boolean equation system. Any causal program has a cycle free equation system. As already said, causality can be only check on the overall program, because two cycle free equation systems could yield a cyclic global system. Indeed, in each reaction, constructive semantics decides which signals are present or absent, then propagates the information and computes a total order. However, variable dependencies in an equation system are only partial orders. In [4], [15], we proposed a technique to sort 4-valued equation systems with respect to their partial orders, which allows to merge two equation systems and to deduce the overall ordering from the previously computed ordering for each system. Doing that, the compilation mechanism falls in two phases: first, a phase where sorted equation systems are computed in a modular way (applying the semantics rules). During this phase we generate sorted 4-valued algebra equation systems. Unlike usual constructive semantics, we don't decide that $\bot$ becomes 0 at this level [2]. On another hand, if a signal has status 0 in an equation system and status 1 in another, the unification must compute $\top$ (error) as resulting status. Thus, a second phase is needed to generate final output code. This phase ensures that as soon as a variable that can be $\top$ appears, the compilation fails. Otherwise, $\bot$ status are changed to 0 and the values are propagated as usual in a Boolean equation system. Phase 2 is the very last stage of compilation and is not done at each reaction. To achieve phase 2, we need an operator called *finalization* (FL for short) not defined for $\top$ and which transform $\bot$ into 0:

x	$FL(x)$
1	1
0	0
$\top$	-
$\bot$	0

1) 4-valued Algebras Candidates: We can consider several algebras to handle the 4-valued signals: we study five possibilities which seem relevant for our purpose.

LE2008: Historically, LE2008 was the first 4-valued algebra we defined in [4], [15]. We designed an imperative synchronous language (*Light Esterel*, LE for short) whose compiler applies the rules of a LE2008 algebra based semantics. In this algebra, only $\boxplus$, $\boxdot$and $\neg$ are considered. $\boxplus$realizes both the unification operation and the Boolean $+$ one. The $\neg$ operator inverses 0 and 1 on one part, and $\bot$ and $\top$ on the other part. Thus we cannot ensure that the function representing the LE2008 equation system associated to a program is monotonic since the operator $\neg$ is not.

Others Algebras: In the other algebras we study, $\sqcup, \sqcap$ and $\neg$ are common to all versions[3]. Intuitively, $\sqcup$performs an unification of the knowledge concerning the signal status according to the interpretation aforementioned: $\bot$ undefined, $\top$ error, 0 absent and 1 present. For instance, if we want to unify status 0 with status 1, the result is error since a signal cannot be both absent and present. Then $\sqcap$is defined to be the dual operator of $\sqcup$. $\neg$ differs from LE2008 definition because the inversions $\neg\bot = \top$ and $\neg\top = \bot$ do not reflect any coherency. Indeed, the negation of error cannot be undefined. Thus, in the algebras we will consider, there is no choice to define the negation: it should inverse 0 and 1 for Boolean coherency, but it should be identity for $\bot$ and $\top$.

Common

$\sqcup$	1	0	$\top$	$\bot$
1	1	$\top$	$\top$	1
0	$\top$	0	$\top$	0
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	1	0	$\top$	$\bot$

$\sqcap$	1	0	$\top$	$\bot$
1	1	$\bot$	1	$\bot$
0	$\bot$	0	0	$\bot$
$\top$	1	0	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

x	$\neg x$
1	0
0	1
$\top$	$\top$
$\bot$	$\bot$

Algebra2

$\boxplus$	1	0	$\top$	$\bot$
1	1	1	$\top$	$\bot$
0	1	0	$\top$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\top$	$\bot$

$\boxdot$	1	0	$\top$	$\bot$
1	1	0	$\top$	$\bot$
0	0	0	$\top$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\top$	$\bot$

The first alternative we study (called Algebra2) is a natural generalization of classical Boolean algebra. Moreover, this algebra propagates the error status ($\top$), which is an interesting feature from compilation point of view.

[2] because an undefined status can increase either to present or to absent in the unification operation

[3] Perhaps $\sqcap$ is useless to compilation concern, but it is the dual operator of $\sqcup$ operator and so is important to prove distributivity properties of operators.

The third algebra we studied (Algebra3) differs from Algebra2 by the behavior of $\perp$: in our view $\perp$ can become 0 or 1. So we must have $1 \boxplus \perp = 1$ (because if $\perp$ becomes 1 or 0, we want the sum results in 1). But, it is not the case for $0 \boxplus \perp$ which only have the possibility to become 1 if $\perp$ becomes 1. Thus, the only changes with regard to Algebra2 are $0 \boxplus \perp = \perp$ and $\perp \boxdot 0 = 0$.

Then we consider another algebra (Algebra4) in which each binary operator has a specific absorbing element ($\top$ for $\sqcup$, $\perp$ for $\sqcap$, 1 for $\boxplus$ and 0 for $\boxdot$). The changes with regard to Algebra2 are $1 \boxplus x = 1$ and $0 \boxdot x = 0$ for any $x = \perp, 0, 1, \top$. This absorption property, which Boolean algebras have, is important to simplify expressions but Algebra4 does not propagate error status.

All these new algebras have not distributivity properties (see table II) and to compute expressions and equation systems (valued in these algebras) solutions, distributivity is a nice property. Hence, we study a last algebra: Algebra5.

Algebra5

$\boxplus$	1	0	$\top$	$\perp$
1	1	1	1	1
0	1	0	$\top$	$\perp$
$\top$	1	$\top$	$\top$	1
$\perp$	1	$\perp$	1	$\perp$

$\boxdot$	1	0	$\top$	$\perp$
1	1	0	$\top$	$\perp$
0	0	0	0	0
$\top$	$\top$	0	$\top$	0
$\perp$	$\perp$	0	0	$\perp$

Algebra5 is variant of Algebra4 where $\top \boxplus \perp$ equals 1 and $\top \boxdot \perp$ equals 0. These modifications could seem strange, but they will make sense in section II-B2 where we will see that they offer the ability to provide Algebra5 with a bilattice structure.

Table II summarizes the logical properties of the algebras defined in section II-B1. Concerning Algebra5, all properties are demonstrated in [8] with both truth table approach and applying algebraic techniques.

2) Bilattice Property of 4-valued Algebra: Indeed, Algebra4 and Algebra5 algebras can be provided with a *bilattice* structure.

Bilattice Theory: Bilattices were introduced by Ginsberg [16] as the underlying framework for AI inference systems. Bilattices are mathematical structures with two distinct orders usually denoted $\leq_B$ and $\leq_K$. $\leq_B$ expresses level of truth and is useful for truth evaluation, $\leq_K$ represents the level of information or knowledge.

Definition 2. (Ginsberg [16]) A **bilattice** is a structure ($\mathscr{B}$, $\leq_B$ $\leq_K$, $\neg$) consisting of a non empty set $\mathscr{B}$, partial orderings $\leq_B$ and $\leq_K$ and a mapping $\neg : \mathscr{B} \mapsto \mathscr{B}$ such that:

1) ($\mathscr{B}$, $\leq_B$) and ($\mathscr{B}$, $\leq_K$) are complete lattices
2) $x \leq_B y \Rightarrow \neg y \leq_B \neg x$, $\forall x, y \in \mathscr{B}$
3) $x \leq_K y \Rightarrow \neg x \leq_K \neg y$, $\forall x, y \in \mathscr{B}$
4) $\neg\neg x = x$, $\forall x \in \mathscr{B}$

If $\leq_B$ is a lattice ordering, let 0 (resp. 1) denotes the least (resp. upper) element. $x \boxplus y$ (resp, $x \boxdot y$) is the join (resp. meet) of x and y. Similarly , if $\leq_K$ is a lattice ordering, we denotes $\perp$ (resp. $\top$) the least (resp. upper) element. $x \sqcup y$ (resp. $x \sqcap y$) is the join (resp. meet) of x and y.

A bilattice satisfies the *interlacing conditions* if:

Properties	LE2008	Alg.2	Alg.3	Alg.4	Alg.5
$\perp \sqcup x = x$	YES	YES	YES	YES	YES
$\perp \sqcap x = \perp$	-	YES	YES	YES	YES
$1 \boxplus x = 1$	no	no	no	YES	YES
$\perp \boxplus x = x$	YES	no	no	no	no
$0 \boxplus x = x$	no	YES	YES	YES	YES
$\perp \boxdot x = \perp$	YES	no	no	no	no
$0 \boxdot x = 0$	no	no	no	YES	YES
$1 \boxdot x = x$	no	YES	YES	YES	YES
$\top \sqcup x = \top$	YES	YES	YES	YES	YES
$\top \sqcap x = x$	-	YES	YES	YES	YES
$\top \boxplus x = \top$	YES	YES	YES	no	no
$\top \boxdot x = \top$	no	YES	YES	no	no
$\top \boxdot x = x$	YES	no	no	no	no
$(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$	YES	YES	YES	YES	YES
$(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$	YES	YES	YES	YES	YES
$(x \boxplus y) \boxplus z = x \boxplus (y \boxplus z)$	YES	YES	YES	YES	YES
$(x \boxdot y) \boxdot z = x \boxdot (y \boxdot z)$	YES	YES	YES	YES	YES
$(x \boxplus y) \boxdot z = (x \boxdot z) \boxplus (y \boxdot z)$	YES	YES	YES	no	YES
$(x \boxdot y) \boxplus z = (x \boxplus z) \boxdot (y \boxplus z)$	YES	YES	YES	no	YES
$(x \sqcup y) \sqcap z = (x \sqcap z) \sqcup (y \sqcap z)$	-	YES	YES	YES	YES
$(x \sqcap y) \sqcup z = (x \sqcup z) \sqcap (y \sqcup z)$	-	YES	YES	YES	YES
$(x \sqcup y) \boxplus z = (x \boxplus z) \sqcup (y \boxplus z)$	YES	no	no	no	YES
$(x \sqcup y) \boxdot z = (x \boxdot z) \sqcup (y \boxdot z)$	YES	no	no	no	YES
$(x \sqcap y) \boxplus z = (x \boxplus z) \sqcap (y \boxplus z)$	-	no	YES	no	YES
$(x \sqcap y) \boxdot z = (x \boxdot z) \sqcap (y \boxdot z)$	-	no	YES	no	YES
$(x \boxplus y) \sqcup z = x \sqcup z \boxplus y \sqcup z$	YES	no	no	no	YES
$(x \boxplus y) \sqcap z = x \sqcap z \boxplus y \sqcap z$	-	no	no	no	YES
$(x \boxdot y) \sqcup z = x \sqcup z \boxdot y \sqcup z$	YES	no	no	no	YES
$(x \boxdot y) \sqcap z = x \sqcap z \boxdot y \sqcap z$	-	no	no		YES
$x \sqcup x = x$	YES	YES	YES	YES	YES
$x \sqcap x = x$	-	YES	YES	YES	YES
$x \boxplus x = x$	YES	YES	YES	YES	YES
$x \boxdot x = x$	YES	YES	YES	YES	YES
$x \boxdot y \boxplus x = x$	YES	no	no	no	YES
$(x \boxplus y) \boxdot x = x$	YES	no	no	no	YES
$(x \sqcup y) \sqcap x = x$	-	YES	YES	YES	YES
$(x \sqcap y) \sqcup x = x$	-	YES	YES	YES	YES
$(x \boxdot y) \sqcup x = x$	YES	no	no	no	no
$(\neg x \boxdot y) \boxplus x = x \boxplus y$	YES	YES	no	no	no
$(\neg x \boxplus y) \boxdot x = x \boxdot y$	YES	YES	no	no	no
$x \boxdot y \boxplus y \boxdot z \boxplus \neg x \boxdot z = x \boxdot y \boxplus \neg x \boxdot z$	YES	YES	no	no	no
$(x \boxplus y) \boxdot (y \boxplus z) \boxdot (\neg x \boxplus z) = (x \boxplus y) \boxdot (\neg x \boxplus z)$	YES	YES	no	no	no
$\neg(x \boxdot y) = \neg x \boxplus \neg y$	YES	YES	YES	YES	YES
$\neg(x \boxplus y) = \neg x \boxdot \neg y$	YES	YES	YES	YES	YES
$\neg(x \sqcup y) = \neg(x) \sqcup \neg(y)$	YES	YES	YES	YES	YES
$\neg(x \sqcap y) = \neg(x) \sqcap \neg(y)$	-	YES	YES	YES	YES

TABLE II. LOGICAL PROPERTIES OF THE CONSIDERED ALGEBRAS

1) $x \leq_B y \Rightarrow x \sqcup z \leq_B y \sqcup z$ and $x \sqcap z \leq_B y \sqcap z$
2) $x \leq_K y \Rightarrow x \boxplus z \leq_K y \boxplus z$ and $x \boxdot z \leq_K y \boxdot z$

In other words, a bilattice is *interlaced* if the lattice operations of one ordering are monotonic with respect to the other ordering and vice versa.

A bilattice is *distributive* if all twelve distributivity laws associated with the 4 operations $\boxplus$, $\boxdot$, $\sqcup$ and $\sqcap$ hold. All distributive bilattice are interlaced [13].

Application to 4-valued Algebras: Bilattice structure is a framework well suited to our concern, since it allows to separate two orderings (Boolean and knowledge) and also

$\bot$	$\leq_K$	0	$\leq_K$	$\top$
$\bot$	$\leq_K$	1	$\leq_K$	$\top$
0	$\leq_B$	$\bot$	$\leq_B$	1
0	$\leq_B$	$\top$	$\leq_B$	1

TABLE III. BILATTICE ORDERS DEFINITION

to still be able to compute equation systems solutions as fixpoints. According to bilattice formalism, we introduce the orders described in table III and we study their properties (see table IV) in the different algebras we have considered. LE2008 algebra, historically, has only the $\leq_B$ ordering.

Properties	Alg.1	Alg.2	Alg.3	Alg.4	Alg.5
$x \leq_K (x \sqcup y)$	-	YES	YES	YES	YES
$x \leq_K y \Rightarrow (x \sqcup z) \leq_K (y \sqcup z)$	-	YES	YES	YES	YES
$x \leq_K y \Rightarrow (x \sqcap z) \leq_K (y \sqcap z)$	-	YES	YES	YES	YES
$\bot \leq_K (x \sqcup y) \leq_K \top$	-	YES	YES	YES	YES
$\bot \leq_K (x \sqcap y) \leq_K \top$	-	YES	YES	YES	YES
$x \leq_K y \Rightarrow \neg x \leq_K \neg y$	-	YES	YES	YES	YES
$x \leq_B (x \boxplus y)$	no	no	no	YES	YES
$(x \boxdot y) \leq_B x$	no	no	no	YES	YES
$x \leq_B y \Rightarrow (x \boxplus z) \leq_B (y \boxplus z)$	YES	no	no	YES	YES
$x \leq_B y \Rightarrow (x \boxdot z) \leq_B (y \boxdot z)$	YES	no	no	YES	YES
$0 \leq_B (x \boxplus y) \leq_B 1$	YES	YES	YES	YES	YES
$0 \leq_B (x \boxdot y) \leq_B 1$	YES	YES	YES	YES	YES
$x \leq_B y \Rightarrow \neg y \leq_B \neg x$	YES	YES	YES	YES	YES
$x \leq_B y$ and $z \leq_B t \Rightarrow x \sqcup z \leq_B y \sqcup t$	YES	YES	YES	YES	YES
$x \leq_B y$ and $z \leq_B t \Rightarrow x \sqcap z \leq_B y \sqcap t$	-	YES	YES	YES	YES
$x \leq_K y$ and $z \leq_K t \Rightarrow x \boxplus z \leq_K y \boxplus t$	-	YES	YES	no	YES
$x \leq_K y$ and $z \leq_K t \Rightarrow x \boxdot z \leq_K y \boxdot t$	-	YES	YES	no	YES

TABLE IV. $\leq_B$ AND $\leq_K$ PROPERTIES IN LE2008 (CALLED ALG.1) AND ALGEBRA(2,3,4,5).FOR ALGEBRA5 ALL LAWS ARE PROVED WITH BOTH TRUTH TABLES AND PURE ALGEBRAIC APPROACH IN [8].

Let us denote $\xi = \{\bot, 0, 1, \top\}$. Algebra(4,5) can be seen as the bilattices (ξ, $\leq_B$, $\leq_K$, $\neg$) according to the previous definition of $\leq_B$ and $\leq_K$ orderings. But Algebra(2,3) cannot. Indeed, (ξ, $\leq_B$) is a complete lattice with 0 and 1 as extremums and so is (ξ, $\leq_K$) with $\bot$ and $\top$ as extremums for Algebra(4,5). But, looking at table IV, we can see that (ξ, $\leq_B$) is not a lattice for Algebra(2,3) because $\boxplus$ and $\boxdot$ are non monotonic operators. According to definition of $\neg$ operator (see section II-B1), for Algebra(4,5) we have: $x \leq_B y \Rightarrow \neg y \leq_B \neg x$, since only 0 and 1 are comparable with respect to $\leq_B$ ordering; $x \leq_K y \Rightarrow \neg x \leq_K \neg y$, since only $\bot$ and $\top$ are comparable with respect to $\leq_K$ ordering; $\neg\neg x = x$. Thus, Algebra(4,5) are bilattices.

In Algebra(4,5), the negation preserves the $\leq_K$ order. This is the expression that $\leq_K$ helps us to characterize the different degrees in the knowledge about element of the algebra. Hence, while it is expected that negation invert the notion of truth from a Boolean point of view, the role of negation with respect to $\leq_K$ is somewhat transparent, we know no more and no less about x than about $\neg x$. As a consequence, Algebra(4,5) with this bilattice structure suits well our concern. We want to find out the appropriate mathematical framework such that algebra equation system solutions can be computed with a fixpoint. In particular, the separation between Boolean consideration and knowledge one is fundamental to make our approach works.

$(x \sqcup y)_h = x_h + y_h$ $(x \sqcup y)_l = x_l + y_l$	$(x \sqcap y)_h = x_h . y_h$ $(x \sqcap y)_l = x_l . y_l$
$(x \boxplus y)_h = x_h + y_h$ $(x \boxplus y)_l = x_l . y_l$	$(x \boxdot y)_h = x_h . y_h$ $(x \boxdot y)_l = x_l + y_l$
$(\neg x)_h = x_l$ $(\neg x)_l = x_h$	

TABLE V. ENCODING RULES FOR ALGEBRA5 OPERATORS WITH RESPECT TO CODING3

Fixpoint computation refines the status of signal from $\bot$ to $\top$ according to $\leq_K$ ordering, so we are interested by monotony only with respect to knowledge order. Nevertheless, the $\leq_B$ order is also mandatory to be able to compute Boolean like ($\boxdot$, $\boxplus$) operations on signal status which can appear in equations.

Distributivity is an important property in bilattice theory in general, but in our case, it is really significant because we can apply these laws to solve 4-valued algebra equation systems. The only algebra where the twelve distributive laws hold is Algebra5 (see table II). So, we can consider that Algebra5 is a distributive bilattice and is the only one among the algebras we consider. We will see in section III the importance of distributive bilattice structure for our algebra.

III. ENCODING BILATTICE INTO BOOLEAN ALGEBRA PRODUCT

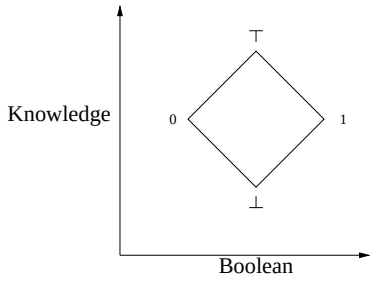

Fig. 1. 4-valued algebras $\leq_B$ and $\leq_K$ orders

For implementation purpose, we are interested to represent element of ξ by Boolean pairs.

A. Encoding

We define encoding functions $e : \xi \mapsto \mathbb{B} \times \mathbb{B}$: $x \in \xi$, $e(x) = (x_h, x_l)$. Several solutions exist and affect the encoding of each operator especially. We sum up 3 possible encoding in the following table. Any other encoding can be deduced by permutation and give equivalent solutions:

Symbol	coding 1	coding 2	coding 3
$\bot$	00	00	00
0	11	10	01
1	01	11	10
$\top$	10	01	11

Coding 2 has the advantage to give a simple operation of *finalization*, it leads to forget x_h component [4]. Coding 3

[4] $FL(x) = x_l$ for coding 2, according to finalization definition

explains the growing of the knowledge: $(0,0)$ unknown, $(0,1)$ or $(1,0)$ good knowledge and $(1,1)$ over-knowledge.

As already said, Algebra5 is a distributive bilattice and seems to be a good candidate to support the definition of synchronous language semantics. However, to really determine which algebra fits our concern better, we study which algebra offers the more efficient encoding into Boolean pairs. We first analyze the encoding of the 5 operators of Algebra(4,5) with respect to coding1, coding2 and coding3. We conclude that it is coding3 which supplies the simpler decomposition into pairs of Boolean for Algebra5. Moreover, for this encoding, the *finalization* operation is as simple as for coding2. It consists in forgetting the x_l component. The results of this study are detailed in [8]. Table V summarizes the effect of coding3 on Algebra5 operators. It confirms that Algebra5 is the best candidate for our purpose. Indeed, Algebra5 offers nice algebraic properties which we will study in the next section.

B. Distributive Bilattice Properties Applied to Algebra5

If we consider Algebra5 and the third encoding (coding 3), we can apply popular results from bilattice theory:

Definition 3. Let $(L, \leq)$ be a complete lattice. The structure $L \bigodot L = (L \times L, \leq_B, \leq_K, \neg)$ is defined as follows:

$$\begin{array}{lll} (x_1,x_2) \leq_B (y_1,y_2) & \text{iff} & x_1 \leq y_1 \text{ and } y_2 \leq x_2 \\ (x_1,x_2) \leq_K (y_1,y_2) & \text{iff} & x_1 \leq y_1 \text{ and } x_2 \leq y_2 \\ \neg(x_1,x_2) = (x_2,x_1) & & \end{array}$$

Lemma 1. (Ginsberg [16]) Let $(L, \leq)$ be a complete lattice. Then $L \bigodot L$ is an interlaced bilattice. If L is distributive, then so is $L \bigodot L$.

Given the structure $L \bigodot L$, it is easy to verify [17] that the basic bilattice operations are defined as follows [5]:

$$\begin{array}{lcl} (c_1,d_1) \sqcup (c_2,d_2) & = & (c_1 + c_2, d_1 + d_2) \\ (c_1,d_1) \sqcap (c_2,d_2) & = & (c_1.c_2, d_1.d_2) \\ (c_1,d_1) \boxplus (c_2,d_2) & = & (c_1 + c_2, d_1.d_2) \\ (c_1,d_1) \boxdot (c_2,d_2) & = & (c_1.c_2, d_1 + d_2) \end{array}$$

$\bigodot$ operation offers a means to built distributive bilattice. We will see that we can construct the algebra we want following this method. Let us consider the usual Boolean set $(\mathbb{B})$. $(\mathbb{B}, \leq)$ is a complete lattice for $0 \leq 1$ ordering:

Property 1. Algebra5 and $\mathbb{B} \bigodot \mathbb{B}$ are isomorphic.

Proof: We recall that Algebra5 is the bilattice (ξ, $\leq_K$ $\leq_B$, $\neg$) defined in section II-B1. Let us consider the encoding $e_3 : \xi \mapsto \mathbb{B} \times \mathbb{B}$ defines as follows: $\bot \mapsto (0,0)$;$0 \mapsto (0,1)$;$1 \mapsto (1,0)$;$\top \mapsto (1,1)$. e_3 is an isomorphism from Algebra5 to $\mathbb{B} \bigodot \mathbb{B}$:
$e_3(x \sqcup y) = e_3(x) \sqcup e_3(y)$:

1) $x = \bot$: $\forall y \in \xi, x \sqcup y = y$. On the other hand $e_3(x) = (0,0)$ then $\forall z \in \mathbb{B} \times \mathbb{B}, e_3(x) \leq_K z$ and then $e_3(x) \sqcup z = z$ and in particular for $e_3(y)$.
2) $x = 0$: the proof falls in two: (1) $y = \bot$ or $y = 0$, $0 \sqcup y = 0$ and $e_3(0 \sqcup y) = (0,1)$, on the other hand, $e_3(0) = (0,1)$ and $e_3(\bot) = (0,0)$ so if $y = \bot$ or $y = 0$, $e_3(0) \sqcup e_3(y) = (0,1)$; (2) $y = 1$ or $y = \top$, $0 \sqcup y = \top$ and $e_3(0 \sqcup y) = (1,1)$, on the other hand, $e_3(1) = (1,0)$ and $e_3(\top) = (1,1)$ so $e_3(0) \sqcup e_3(y) = (1,1)$.
3) $x = 1$ the proof is similar to the previous case.
4) $x = \top$, $\forall y \in \xi, x \sqcup y = x$. On the other hand, $e_3(x) = (1,1)$ then $\forall z \in \mathbb{B} \times \mathbb{B}, z \leq_K e_3(x)z$ and in particular $e_3(y)$ so $e_3(x) \sqcup e_3(y) = e_3(x)$.

The proofs for $e_3(x \sqcap y) = e_3(x) \sqcap e_3(y)$, $e_3(x \boxplus y) = e_3(x) \boxplus e_3(y)$ and $e_3(x \boxdot y) = e_3(x) \boxdot e_3(y)$ are similar to the studied case and detail in length in [8].

Finally, $e_3(\neg x) = \neg(e_3(x))$: if $x = \bot$ or $x = \top$ the result is immediate since we have $e_3(x) = (x_h, x_l)$ with $x_h = x_l$ [6], hence $(x_h, x_l) = (x_l, x_h)$. $\neg 0 = 1$ hence $e_3(\neg 0) = (1,0) = \neg(0,1) = \neg(e_3(0))$. The proof for $x = 1$ is symmetric.

Moreover, e_3 is clearly a bijection. ■

As a consequence, each Algebra5 equation system is equivalent to a Boolean equation system where each equation $x = f(\vec{v})$ is expanded in two equations: $x_h = f(\vec{v})_h$ and $x_l = f(\vec{v})_l$ such that $(f(\vec{v})_h, f(\vec{v})_l) = e_3(f(\vec{v}))$.

We applied this work in the CLEM toolkit design. The semantics of the language is now defined in Algebra5 framework. To compile programs we implement the semantics rules and thanks to property 1, we can project 4-valued equations into pair of Boolean ones. Hence, the compiler computes a Boolean equation system for each correct program. From these equation systems, we generate software code as well as hardware implementation or model-checking input.

IV. Related Works

First of all, we cannot discuss related works without citing K.Schneider, J.Brandt and T.Schuele [3]. These authors have defined the synchronous language `Quartz` and explained the underlying theoretical aspects of the compilation. In `Quartz`, they needed to handle 4-valued equations with three operators. To check Quartz program's causality, they compute least and greatest fixpoints. Thus, they had to define an order relation and they took the knowledge order. With this order, they refined the language operators, defined two new $inf()$ and $sup()$ operators and finally found the $\boxplus$, $\boxdot$, $\neg$, $\sqcup$ and $\sqcap$ of the Ginsberg bilattice! They also took the encoding of the Ginsberg bilattice too. Based on and according to Bryant work about the BDDs, they are convinced that this encoding (called Dual Rail Encoding) is the best encoding and took it in their compiler. Authors never refer Ginsberg bilattice approach and don't benefit from all the power of this 4-valued algebra (in particular, the 12 distributive laws specific to this bilattice). Indeed, Schneider and others use this approach mainly to check Quartz programs causality. On our side, bilattice structure and operators provide us with an efficient means to compute signal environments. Nevertheless, their choice strengthens our opinion concerning Algebra5 as a well suited model to define synchronous language semantics.

[5] we denote the meet and join operations with respect to $\leq_B$ order by $\boxdot$ and $\boxplus$; the meet and join operations with respect to $\leq_K$ order by $\sqcup$ and $\sqcap$ and the meet and join operation in the underlying lattice L by $+$ and $.$

[6] $e_3(\bot) = (0,0)$ and $e_3(\top) = (1,1)$

Usually, multi-valued algebras are very often use in model-checker, symbolic Boolean exploration of state space and they intervene in mapping circuit on FPGA and PLD. Multi-valued algebras have lots of applications in design and verification of switching circuit. For instance, ten-value logic or more [18] are used to generate test patterns. Similarly, Dubrova [19] gives an overview concerning MOS technology considerations as well as memories synthesis or arithmetic circuits verification. Moreover a hardware synthesis language as VHDL specified in the IEEE 1164, defines two relevant algebras for built systems: MVL5 (resp. MVL9) based on `U,0,1,Z,X` symbols (resp.`U,X,0,1,Z,W,L,H,-` symbols). However, the multi-valued approach can have larger applications. For example, V. D. Shet [20] dedicated his PHD thesis to the application of bilattices to people visual surveillance.

V. Conclusion and Future Works

Synchronous languages have formal semantics computing models of programs either for verification purpose or for compilation. All along the three last decades, several semantics have been defined. They are all in common to compute the status of signals in an execution of a program. We need a mathematical framework to represent and compute signal status according to semantics rules. This paper is a review of some adopted solutions and points out another framework which provides us with both verification and separated compilation. It studies classical approaches with 3-valuated algebras. In these algebras, a $\bot$ element turns out to be useful to have the ability to apply constructive rules. But to go further and get a separated compilation means relying on semantics, 4-valued algebras are required. We study five different 4-valued algebras and show that Algebra5 is a distributive bilattice. Then, we can consider two orders: a Boolean order and a knowledge order which allows us to rely on fixpoints computation to establish signal status. The Boolean order is useful to compute the current environment of signals and the knowledge order is essential to merge environments after a separated compilation of statements. Nevertheless, Algebra3 is also an appealing framework because error is propagated which is not the case for Algebra5. But as soon as $\top$ becomes an absorbing element, the bilattice structure cannot exits and this property is really inescapable.

The motivation for this work is the definition of a new method to compile synchronous languages in a separated way (see [4]). Algebra5 provides us with well-suited properties allowing us to define a constructive and efficient semantics. It is also important to specify a behavioral semantics. It gives a meaning to programs and generates models for verification purpose. The chosen framework allows to express these two semantics and their equivalence holds and is immediate. Moreover, the isomorphism between Algebra5 and $\mathbb{B}\bigodot\mathbb{B}$ and the chosen encoding allows us to compute solutions as Boolean equation system solutions. Our previous work considered LE2008 algebra as foundation of the semantics. The advantages were its Boolean algebra structure and the simplicity of the finalization process. The drawback was the $\neg$ operator interpretation incompatible with classical Boolean interpretation. Algebra5 is not a Boolean algebra. However, thanks to its bilattice structure, the most important laws, relevant compute equation system solutions (commutativity, associativity, distributivity, neutral elements and De Morgan's laws), hold. Moreover, the integration in our compiler was immediate. To switch from LE2008 to Algebra5, we only needed to implement again the projection rules defined in section III-A, and telling us how Algebra5 operators are defined as Boolean operators.

References

[1] G. Berry, *The Constructive Semantics of Pure Esterel*. Draft Book, available at: http://www.esterel-technologies.com 1996.

[2] G. Plotkin, "A structural approach to operational semantics," DAIMI FN-19, Aarhus University, Denmark, Tech. Rep., 1981.

[3] K. Schneider, J. Brandt, and T. Schuele, "Causality analysis of synchronous programs with delayed actions," in *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, ser. CASES '04. New York, NY, USA: ACM, 2004, pp. 179–189. [Online]. Available: http://doi.acm.org/10.1145/1023833.1023859

[4] A. Ressouche, D. Gaffé, and V. Roy, "Modular compilation of a synchronous language," in *Software Engineering Research, Management and Applications*, ser. Studies in Computational Intelligence, R. Lee, Ed. Springer, Aug. 2008, vol. 150, pp. 151–171. [Online]. Available: http://hal.inria.fr/inria-00523528

[5] D. Gaffé and A. Ressouche, "The clem toolkit," in *Proceedings of 23rd IEEE/Aberry-bookCM International Conference on Automated Software Engineering(ASE 2008)*, L'aquila, Italy, September 2008.

[6] A. Benveniste, P. L. Guernic, and C. Jacquemot, "Synchronous programming with events and relations : the signal language and its semantics," *Science of computer programming*, vol. 16, no. 2, pp. 103–149, 1991.

[7] E. Post, "Introduction to a general theory of elementary propositions," *Amer. J.Math.*, vol. 43, pp. 163–185, 1921.

[8] D. Gaffé and A. Ressouche, "Algebras and Synchronous Language Semantics," INRIA, Rapport de recherche RR-8138, Nov. 2012. [Online]. Available: http://hal.inria.fr/hal-00752976

[9] D. Bochvar, "On a three-valued logical calculus and its applications to the analysis of the paradoxes of the classical extended functional calculus (in russian)," *Matématičeskij Sbornik*, vol. 4, pp. 287–308, 1938, translated by Merrie Bergman, History and Philosophy of Logic 2, (1981), 87–112.

[10] J. Łukasiewicz, "O logice tròjwartościowej," *Ruch Filozoficzny*, vol. 5, pp. 169–171, 1920, reprinted and translated in *Jan Łukasiewicz, Selected Writings*.

[11] D. S. Scott, "Logic and programming languages," *Communications of the ACM*, vol. 20, pp. 634–641, 1977.

[12] S. C. Kleene, *Introduction to metamathematics*, ser. Bibl. Matematica. Amsterdam: North-Holland, 1952.

[13] N. Belnap, "A useful four-valued logic," *Modern Uses of Multiple-Valued Logic*, pp. 3–37, 1977.

[14] C. Huizing and R. Gerth, "Semantics of reactive systems in abstract time," in *Real Time: Theory in Practice, Proc of REX workshop*. W.P. de Roever and G. Rozenberg Eds,LNCS, June 1991, pp. 291–314.

[15] A. Ressouche and D. Gaffé, "Compilation modulaire d'un langage synchrone," *Revue des sciences et technologies de l'information, série Théorie et Science Informatique*, vol. 4, no. 30, pp. 441–471, Jun. 2011. [Online]. Available: http://hal.inria.fr/inria-00524499

[16] M. Ginsberg, "Multivalued logics: A uniform approach to inference in artificial intelligence," *Computational Intelligence*, vol. 4, pp. 265–316, 1988.

[17] D. P. B. Mobasher and G. Slutzki, "Multi-valued logic programming semantics: An algebraic approach," *Theoretical Computer Science*, vol. 171 (1-2), pp. 77–109, 1997.

[18] S. Bose, P. Agrawal, and V. D. Agrawal, "A path delay fault simulator for sequential circuits," in *VLSI Design*, 1993, pp. 269–274.

[19] E. Dubrova, "Multiple-valued logic in vlsi: Challenges and opportunities," in *In Proceedings of NORCHIP'99*, 1999.

[20] V. D. Shet, "Bilattice based logical reasoning for automated visual surveillance and other applications," Ph.D. dissertation, University of Maryland College Park, 2007.